Workflow fairness control on online and non-clairvoyant distributed computing platforms

Rafael Ferreira da Silva, Tristan Glatard, Frédéric Desprez

Workflow fairness control on online and non-clairvoyant distributed computing platforms

Rafael Ferreira da Silva[1], Tristan Glatard[1] and Frédéric Desprez[2]

[1] University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
{rafael.silva,glatard}@creatis.insa-lyon.fr
[2] INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France
Frederic.Desprez@inria.fr

Abstract. Fairly allocating distributed computing resources among workflow executions is critical to multi-user platforms. However, this problem remains mostly studied in clairvoyant and offline conditions, where task durations on resources are known, or the workload and available resources do not vary along time. We consider a non-clairvoyant, online fairness problem where the platform workload, task costs and resource characteristics are unknown and not stationary. We propose a fairness control loop which assigns task priorities based on the fraction of pending work in the workflows. Workflow characteristics and performance on the target resources are estimated progressively, as information becomes available during the execution. Our method is implemented and evaluated on 4 different applications executed in production conditions on the European Grid Infrastructure. Results show that our technique reduces slowdown variability by 3 to 7 compared to first-come-first-served.

1 Introduction

The problem of fairly allocating computing resources to application workflows rapidly arises on shared computing platforms such as grids or clouds. It must be addressed whenever the demand for resources is higher than the offer, that is, when some workflows are slowed down by concurrent executions. In some cases, unfairness makes the platform totally unusable, for instance when very short executions are launched concurrently with longer ones. We define fairness as in [1,2,3], i.e. as the variability in a set of workflows of the *slowdown* $\frac{M_{multi}}{M_{own}}$, where M_{multi} is the makespan when concurrent executions are present, and M_{own} is the makespan without concurrent executions.

We consider a software-as-a-service platform where users can, at any time, launch application workflows that will compete for computing resources. Our two main assumptions are (i) that the problem is *online*: new workflows can be submitted at any time, and resources may also join or leave at any time, and (ii) that the problem is *non-clairvoyant*: the execution time of a task on a given computing resource is unknown. Non-clairvoyance comes from the lack of application models in the platform and from the lack of information about the performance of computing and network resources. We also assume a limited control on the scheduler, i.e. that only task priorities can be changed to influence scheduling.

These conditions are representative of a large set of platforms, for instance the Virtual Imaging Platform (VIP [4]) and other science gateways [5,6,7] deployed on the European Grid Infrastructure (EGI[3]). These gateways offer applications deployed as workflows on shared computing platforms, but they have no information about when users will launch them and how long each task will last on a given resource.

Fairness among workflow executions has been addressed in several studies which, however, mostly assume clairvoyant conditions. For instance, the works in [2,1,3,8,9,10] either directly minimize the slowdown (which assumes that makespans can be predicted) or use heuristics assuming that task durations and resources are known. A notable exception is found in [11], where a non-clairvoyant algorithm is proposed: nevertheless, it is purely offline, assuming that the tasks and resources are known and do not vary.

In this work, we propose an algorithm to control fairness on non-clairvoyant online platforms. Based on a progressive discovery of applications' characteristics on the infrastructure, our method dynamically estimates the fraction of pending work for each workflow. Task priorities are then adjusted to harmonize this fraction among active workflows. This way, resources are allocated to application workflows relatively to their amount of work to compute. The method is implemented in VIP, and evaluated with different workflows, in production conditions, on the EGI. We use the slowdown as a *post-mortem* metric, to evaluate our method once execution times are known. Contributions of this paper are:

1. A new instantiation of our control loop [12] to handle unfairness, consisting of (i) an online, non-clairvoyant fairness metric, and (ii) a task prioritization algorithm.
2. Experiments demonstrating that this method improves fairness compared to a first-come-first-served approach, in production conditions, and using 4 different applications.

The next section details our fairness control process, and section 3 presents experiments and results.

2 Fairness control process

Workflows are directed graphs of activities spawning sequential tasks for which the executable and input data are known, but the computational cost and produced data volume are not. Workflow graphs may include conditional and loop operators . Algorithm 1 summarizes our fairness control process. Fairness is controlled by allocating resources to workflows according to their fraction of pending work. It is done by re-prioritising tasks in workflows where the unfairness degree η_u is greater than a threshold τ_u. This section describes how η_u and τ_u are computed, and details the re-prioritization algorithm.

Measuring unfairness: η_u. Let m be the number of workflows with an active activity; a workflow activity is active if it has at least one waiting (queued) or

[3] `http://www.egi.eu`

Algorithm 1 Main loop for fairness control

```
1: input: m workflow executions
2: while there is an active workflow do
3:     wait for timeout or task status change in any workflow
4:     determine unfairness degree η_u
5:     if η_u > τ_u then
6:         re-prioritize tasks using Algorithm 2
7:     end if
8: end while
```

running task. The unfairness degree η_u is the maximum difference between the fractions of pending work:

$$\eta_u = W_{\max} - W_{\min}, \tag{1}$$

with $W_{\min} = \min\{W_i, i \in [1, m]\}$ and $W_{\max} = \max\{W_i, i \in [1, m]\}$. All W_i are in $[0, 1]$. For $\eta_u = 0$, we consider that resources are fairly distributed among all workflows; otherwise, some workflows consume more resources than they should. The fraction of pending work W_i of a workflow $i \in [1, m]$ is defined from the fraction of pending work $w_{i,j}$ of its n_i active activities:

$$W_i = \max_{j \in [1, n_i]} (w_{i,j}) \tag{2}$$

All $w_{i,j}$ are between 0 and 1. A high $w_{i,j}$ value indicates that the activity has a lot of pending work compared to the others. We define $w_{i,j}$ as:

$$w_{i,j} = \frac{Q_{i,j}}{Q_{i,j} + R_{i,j} P_{i,j}} \cdot \hat{T}_{i,j}, \tag{3}$$

where $Q_{i,j}$ is the number of waiting tasks in the activity, $R_{i,j}$ is the number of running tasks in the activity, $P_{i,j}$ is the performance of the activity, and $\hat{T}_{i,j}$ is its relative observed duration. $\hat{T}_{i,j}$ is defined as the ratio between the median duration $\tilde{t}_{i,j}$ of the completed tasks in activity j and the maximum median task duration among all active activities of all running workflows:

$$\hat{T}_{i,j} = \frac{\tilde{t}_{i,j}}{\max_{v \in [1,m], w \in [1, n_i^*]} (\tilde{t}_{v,w})} \tag{4}$$

Tasks of an activity all consist of the following successive phases: `setup`, `inputs download`, `application execution` and `outputs upload`; $\tilde{t}_{i,j}$ is computed as $\tilde{t}_{i,j} = \tilde{t}_{i,j}^{setup} + \tilde{t}_{i,j}^{input} + \tilde{t}_{i,j}^{exec} + \tilde{t}_{i,j}^{output}$. Medians are progressively estimated as tasks complete. At the beginning of the execution, $\hat{T}_{i,j}$ is initialized to 1 and all medians are undefined; when two tasks of activity j complete, $\tilde{t}_{i,j}$ is updated and $\hat{T}_{i,j}$ is computed with equation 4. In this equation, the max operator is computed only on $n_i^* \leq n_i$ activities with at least 2 completed tasks, i.e. for which $\tilde{t}_{i,j}$ can be determined. We are aware that using the median may be inaccurate. However, without a model of the applications' execution time, we must rely on observed task durations. Using the whole time distribution (or at least its few first moments) may be more accurate but it would complexify the method.

In Eq. 3, the performance $P_{i,j}$ of an activity varies between 0 and 1. A low $P_{i,j}$ indicates that resources allocated to the activity have bad performance for the activity; in this case, the contribution of running tasks is reduced and $w_{i,j}$ increases. Conversely, a high $P_{i,j}$ increases the contribution of running tasks, therefore decreases $w_{i,j}$. For an activity j with k_j active tasks, we define $P_{i,j}$ as:

$$P_{i,j} = 2\left(1 - \max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\}\right), \tag{5}$$

where $t_u = t_u^{setup} + t_u^{input} + t_u^{exec} + t_u^{output}$ is the sum of the estimated durations of task u's phases. Estimated task phase durations are computed as the max between the current elapsed time in the task phase (0 if the task phase has not started) and the median duration of the task phase. $P_{i,j}$ is initialized to 1, and updated using Eq. 5 only when at least 2 tasks of activity j are completed.

If all tasks perform as the median, i.e. $t_u = \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} = 0.5$ and $P_{i,j} = 1$. Conversely, if a task in the activity is much longer than the median, i.e. $t_u \gg \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} \approx 1$ and $P_{i,j} \approx 0$. This definition of $P_{i,j}$, considers that bad performance results in a few tasks blocking the activity. Indeed, we assume that the scheduler doesn't deliberately favor any activity and that performance discrepancies are manifested by a few "unlucky" tasks slowed down by bad resources. Performance, in this case, has a relative definition: depending on the activity profile, it can correspond to CPU, RAM, network bandwidth, latency, or a combination of those. We admit that this definition of $P_{i,j}$ is a bit rough. However, under our non-clairvoyance assumption, estimating resource performance for the activity more accurately is hardly possible because (i) we have no model of the application, therefore task durations cannot be predicted from CPU, RAM or network characteristics, and (ii) network characteristics and even available RAM are shared among concurrent tasks running on the infrastructure, which makes them hardly measurable.

Thresholding unfairness: τ_u. Task prioritisation is triggered when the unfairness degree is considered critical, i.e $\eta_u > \tau_u$. Thresholding consists in clustering platform configurations in two groups: one for which unfairness is considered acceptable, and one for which task re-prioritization is needed. We determine τ_u from execution traces, for which different thresholding approaches can be used. For instance, we could consider that x% of the platform configurations are unfair while the rest are acceptable. The choice of x, however, would be arbitrary. Instead, we inspect the modes of the distribution of η_u to determine a threshold with a practical justification: values of η_u in the highest mode of the distribution, i.e. which are clearly separated from the others, will be considered unfair.

In this work, the distribution of η_u is measured from traces collected in VIP between January 2011 and April 2012 [13]. The data set contains 680,988 tasks (including resubmissions and replications) of 2,941 workflow executions executed by 112 users; task average queueing time is about 36 min. Applications deployed in VIP are described as GWENDIA workflows [14] executed using the MOTEUR workflow engine [15]. Resource provisioning and task scheduling are provided by

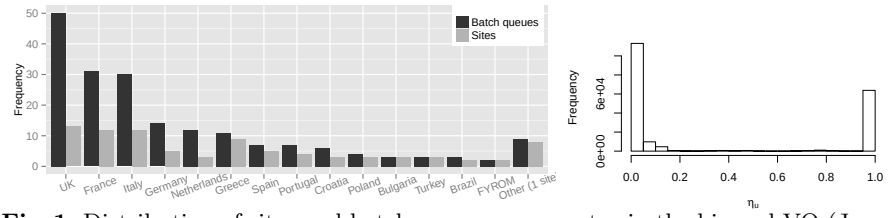


Fig. 1. Distribution of sites and batch queues per country in the biomed VO (January 2013) (*left*) and histogram of the unfairness degree η_u sampled in bins of 0.05 (*right*).

DIRAC [16]. Resources are provisioned online with no advance reservations. Tasks are executed on the biomed virtual organization (VO) of the European Grid Infrastructure (EGI) which has access to some 150 computing sites worldwide and to 120 storage sites providing approximately 4 PB of storage. Fig. 1 (left) shows the distribution of sites per country supporting the biomed VO.

The unfairness degree η_u was computed after each event found in the data set. Fig. 1 (right) shows the histogram of these values, where only $\eta_u \neq 0$ values are represented. This histogram is clearly bi-modal, which is a good property since it reduces the influence of τ_u. From this histogram, we choose $\tau_u = 0.2$. For $\eta_u > 0.2$, task prioritization is triggered.

Task prioritization. The action taken to cope with unfairness is to increase the priority of $\Delta_{i,j}$ waiting tasks for all activities j of workflow i where $w_{i,j} - W_{\min} > \tau_u$. Running tasks cannot be pre-empted. Task priority is an integer initialized to 1. $\Delta_{i,j}$ is determined so that $\tilde{w}_{i,j} = W_{min} + \tau_u$, where $\tilde{w}_{i,j}$ is the estimated value of $w_{i,j}$ after $\Delta_{i,j}$ tasks are prioritized. We approximate $\tilde{w}_{i,j}$ as:

$$\tilde{w}_{i,j} = \frac{Q_{i,j} - \Delta_{i,j}}{Q_{i,j} + R_{i,j}P_{i,j}} \hat{T}_{i,j},$$

which assumes that $\Delta_{i,j}$ tasks will move from status queued to running, and that the performance of new resources will be maximal. It gives:

$$\Delta_{i,j} = Q_{i,j} - \left\lfloor \frac{(\tau_u + W_{\min})(Q_{i,j} + R_{i,j}P_{i,j})}{\hat{T}_{i,j}} \right\rfloor, \tag{6}$$

where $\lfloor\rfloor$ rounds a decimal down to the nearest integer value.

Algorithm 2 describes our task re-prioritization. *maxPriority* is the maximal priority value in all workflows. The priority of $\Delta_{i,j}$ waiting tasks is set to *maxPriority+1* in all activities j of workflows i where $w_{i,j} - W_{min} > \tau_u$. Note that this algorithm takes into account scatter among W_i although η_u ignores it (see Eq. 1). Indeed, tasks are re-prioritized in *any* workflow i for which $W_i - W_{\min} > \tau_u$.

The method also accommodates online conditions. If a new workflow i is submitted, then $R_{i,j} = 0$ for all its activities and $\hat{T}_{i,j}$ is initialized to 1. This leads to $W_{max} = W_i = 1$, which increases η_u. If η_u goes beyond τ_u, then $\Delta_{i,j}$ tasks of activity j of workflow i have their priorities increased to restore fairness. Similarly, if new resources arrive, then $R_{i,j}$ increase and η_u is updated accordingly. Table 1 illustrates the method on a simple example.

3 Experiments and results

Experiments are performed on a production grid platform to ensure realistic conditions. Evaluating fairness in production by measuring the slowdown is not

Algorithm 2 Task re-prioritization

```
1: input: W_1 to W_m //fractions of pending works
2: maxPriority = max task priority in all workflows
3: for i=1 to m do
4:    if W_i − W_min > τ_u then
5:       for j=1 to a_i do
6:          //a_i is the number of active activities in workflow i
7:          if w_{i,j} − W_min > τ_u then
8:             Compute Δ_{i,j} from equation 6
9:             for p=1 to Δ_{i,j} do
10:               if ∃ waiting task q in activity j with priority ≤ maxPriority then
11:                  q.priority = maxPriority + 1
12:               end if
13:            end for
14:         end if
15:      end for
16:   end if
17: end for
```

straightforward because M_{own} (see definition in the introduction) cannot be directly measured. As described in Section 3.1, we estimate the slowdown from task durations, but this estimation may be challenged. Thus, Experiment 1 evaluates our method on a set of identical workflows, where the variability of the measured makespan can be used as a fairness metric. In Experiment 2, we add a very short workflow to this set of identical workflow, which was one of the configurations motivating this study. Finally, Experiment 3 considers the more general case of 4 different workflows with heterogeneous durations.

3.1 Experiment conditions

Fairness control was implemented as a MOTEUR plugin receiving notifications about task and workflow status changes. Each workflow plugin forwards task status changes and $\tilde{t}_{i,j}$ values to a service centralizing information about all the active workflows. This service then re-prioritizes tasks according to Algorithms 1 and 2. As no online task modification is possible in DIRAC, we implemented task prioritization by canceling and resubmitting queued tasks to DIRAC with new priorities. This implementation decision adds an overhead to task executions. Therefore, the timeout value used in Algorithm 1 is set to 3 minutes.

The computing platform for these experiments is the biomed VO used to determine τ_u in Section 2. To ensure resource limitation without flooding the production system, experiments are performed only on 3 sites of different countries (France, Spain and Netherlands). Four real medical simulation workflows are considered: `GATE` [17], `SimuBloch`, `FIELD-II` [18], and `PET-Sorteo` [19]; their main characteristics are summarized on Table 2.

Three experiments are conducted. Experiment 1 tests whether unfairness among *identical workflows* is properly addressed. It consists of three `GATE` workflows sequentially submitted, as users usually do in the platform. Experiment 2 tests if the performance of *very short workflow executions* is improved by the fairness mechanism. Its workflow set has three `GATE` workflows launched sequentially, followed by a `SimuBloch` workflow. Experiment 3 tests whether unfairness among *different workflows* is detected and properly handled. Its workflow

Let's consider two identical workflows composed of one activity with 6 tasks, and assume the following values at time t:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$\hat{T}_{i,1}$	$W_i = w_{i,1}$
1	1	3	10	0.9	1.0	0.27
2	6	0	-	1.0	1.0	1.00

Values unknown at time t are noted '-'. Workflow 1 has 2 completed and 3 running tasks with the following phase durations (in arbitrary time units):

u	t_u^{setup}	t_u^{input}	t_u^{exec}	t_u^{output}	t_u
1	2	2	4	1	9
2	1	2	3	2	8
3	2	3	5	-	-
4	2	2	-	-	-
5	1	-	-	-	-

, We have $\tilde{t}_{1,1}^{setup} = 2$, $\tilde{t}_{1,1}^{input} = 2$, $\tilde{t}_{1,1}^{exec} = 4$ and $\tilde{t}_{1,1}^{output} = 2$. Therefore, $\tilde{t}_{1,1} = 10$.

The configuration is clearly unfair since workflow 2 has not started tasks.
Eq. 1 gives $\eta_u = 0.73$. As $\eta_u > \tau_u = 0.2$, the platform is considered unfair and task re-prioritization is triggered.

$\Delta_{2,1}$ tasks from workflow 2 should be prioritized. According to Eq. 6:
$\Delta_{2,1} = Q_{2,1} - \left\lfloor \frac{(\tau_u + W_1)(Q_{2,1} + R_{2,1} P_{2,1})}{\hat{T}_{2,1}} \right\rfloor = 6 - \left\lfloor \frac{(0.2+0.27)(6+0 \cdot 1.0)}{1.0} \right\rfloor = 4$

At time $t' > t$:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$\hat{T}_{i,1}$	$W_i = w_{i,1}$
1	1	3	10	0.8	1.0	0.29
2	2	4	-	1.0	1.0	0.33

.

Now, $\eta_u = 0.04 < \tau_u$. The platform is considered fair and no action is performed.

Table 1. Example

set consists of a `GATE`, a `FIELD-II`, a `PET-Sorteo` and a `SimuBloch` workflow launched sequentially.

For each experiment, a workflow set using our fairness mechanism (`Fairness` – `F`) is compared to a control workflow set (`No-Fairness` – `NF`). No method from the literature could be included in the comparison because, as mentioned in the introduction, they are either non-clairvoyant or offline. `Fairness` and `No-Fairness` are launched simultaneously to ensure similar grid conditions. For each task priority increase in the `Fairness` workflow set, a task in the `No-Fairness` workflow set task queue is also prioritized to ensure equal race conditions for resource allocation. Experiment results are not influenced by the re-submission process overhead since both `Fairness` and `No-Fairness` experience the same overhead. Four repetitions of each experiment are done, along a time period of four weeks to cover different grid conditions. Grid conditions vary among repetitions because computing, storage and network resources are shared with other users . We use MOTEUR 0.9.21, configured to resubmit failed tasks up to 5 times, and with the task replication mechanism described in [12] activated. We use the DIRAC v6r5p1 instance provided by France-Grilles[4], with a

[4] `https://dirac.france-grilles.fr`

Workflow		#Tasks	CPU time	Input	Output
GATE	(CPU-intensive)	100	few minutes to one hour	∼115 MB	∼40 MB
SimuBloch	(data-intensive)	25	few seconds	∼15 MB	< 5 MB
FIELD-II	(data-intensive)	122	few seconds to 15 minutes	∼208 MB	∼40 KB
PET-Sorteo	(CPU-intensive)	1→80→1→80→1→1	∼10 minutes	∼20 MB	∼50 MB

Table 2. Workflow characteristics (→ indicate task dependencies).

first-come, first-served policy imposed by submitting workflows with decreasing priority values.

Two different fairness metrics are used. The unfairness μ is the area under the curve η_u during the execution:

$$\mu = \sum_{i=2}^{M} \eta_u(t_i) \cdot (t_i - t_{i-1}),$$

where M is the number of time samples until the makespan. This metric measures if the fairness process can indeed minimize its own criterion η_u. In addition, the slowdown s of a completed workflow execution is defined by:

$$s = \frac{M_{multi}}{M_{own}}$$

where M_{multi} is the makespan observed on the shared platform, and M_{own} is the estimated makespan if it was executed alone on the platform. In our conditions, M_{own} is estimated as:

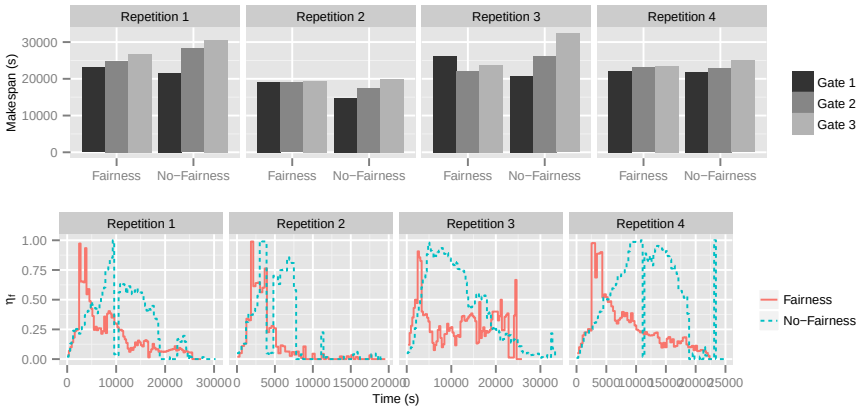
$$M_{own} = \max_{p \in \Omega} \sum_{u \in p} t_u,$$

where Ω is the set of task paths in the workflow, and t_u is the measured duration of task u. This assumes that concurrent executions only impact task waiting time, not performance. For instance, network congestion or changes in performance distribution resulting from concurrent executions are ignored. We use σ_s, the standard deviation of the slowdown to quantify the unfairness. In Experiment 1, the standard deviation of the makespan (σ_m) is also used.

3.2 Results and discussion

Experiment 1 (*identical workflows*): Fig. 2 shows the makespan, unfairness degree η_u, makespan standard deviation σ_m, slowdown standard deviation σ_s and unfairness μ for the 4 repetitions. The difference among makespans and unfairness degree values are significantly reduced in all repetitions of `Fairness`. Both `Fairness` and `No-Fairness` behave similarly until η_u reaches the threshold value $\tau_u = 0.2$. Unfairness is then detected and the mechanism triggers task prioritization. Paradoxically, the first effect of task prioritization is a slight increase of η_u. Indeed, $P_{i,j}$ and $\hat{T}_{i,j}$, that are initialized to 1, start changing earlier in `Fairness` than in `No-Fairness` due to the availability of task duration values to compute $\tilde{t}_{i,j}$. Note that η_u reaches similar maximal values in both cases, but reaches them faster in `Fairness`. The fairness mechanism then manages to decrease η_u back under 0.2 much faster than it happens in `No-Fairness` when tasks progressively complete. Finally, slight increases of η_u are sporadically observed towards the end of the execution. This is due to task replications performed by MOTEUR: when new tasks are created, the fraction of pending work W increases, which has an effect on η_u. Quantitatively, the fairness mechanism reduces σ_m up to a factor of 15, σ_s up to a factor of 7, and μ by about 2.

Experiment 2 (*very short execution*): Fig. 3 shows the makespan, unfairness degree η_u, unfairness μ and slowdown standard deviation. In all cases, the



Repetition 1				Repetition 2				Repetition 3				Repetition 4			
	$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$
NF	4666	1.03	8758	NF	2541	0.50	4154	NF	5791	2.10	13392	NF	1567	0.87	12283
F	1884	0.40	5292	F	167	0.07	2367	F	2007	0.84	7243	F	706	0.24	6070

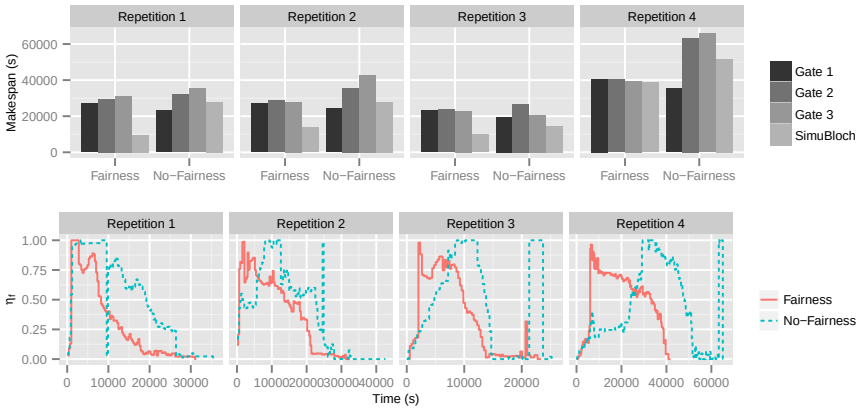
Fig. 2. Experiment 1 (identical workflows). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: makespan standard deviation σ_m, slowdown standard deviation σ_s and unfairness μ.

makespan of the very short `SimuBloch` executions is significantly reduced for `Fairness`. The evolution of η_u is coherent with Experiment 1: a common initialization phase followed by an anticipated growth and decrease for `Fairness`. `Fairness` reduces σ_s up to a factor of 5.9 and unfairness up to a factor of 1.9.

Table 3 shows the execution makespan (m), average wait time ($\bar{w}$) and slowdown (s) values for the `SimuBloch` execution launched after the 3 `GATE`. As it is a non-clairvoyant scenario where no information about task execution time and future task submission is known, the fairness mechanism is not able to give higher priorities to `SimuBloch` tasks in advance. Despite that, the fairness mechanism speeds up `SimuBloch` executions up to a factor of 2.9, reduces task average wait time up to factor of 4.4 and reduces slowdown up to a factor of 5.9.

Experiment 3 (*different workflows*): Fig. 4 shows slowdown, unfairness degree, unfairness μ and slowdown standard deviation σ_s for the 4 repetitions. `Fairness` slows down `GATE` while it speeds up all other workflows. This is because `GATE` is the longest and the first to be submitted; in `No-Fairness`, it is favored by resource allocation to the detriment of other workflows. The evolution of η_u is similar to Experiments 1 and 2. σ_s is reduced up to a factor of 3.8 and unfairness up to a factor of 1.9.

In all 3 experiments, fairness optimization takes time to begin because the method needs to acquire information about the applications which are totally unknown when a workflow is launched. We could think of reducing the time of this information-collecting phase, e.g. by designing initialization strategies maximizing information discovery, but it couldn't be totally removed. Currently, the method works best for applications with a lot of short tasks because the first few tasks can be used for initialization, and optimization can be exploited for the



Repetition 1	σ_s	$\mu(s)$
NF	94.88	17269
F	15.95	9085

Repetition 2	σ_s	$\mu(s)$
NF	100.05	16048
F	42.94	12543

Repetition 3	σ_s	$\mu(s)$
NF	87.93	11331
F	57.62	7721

Repetition 4	σ_s	$\mu(s)$
NF	213.60	28190
F	76.69	21355

Fig. 3. Experiment 2 (very short execution). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: unfairness μ and slowdown standard deviation.

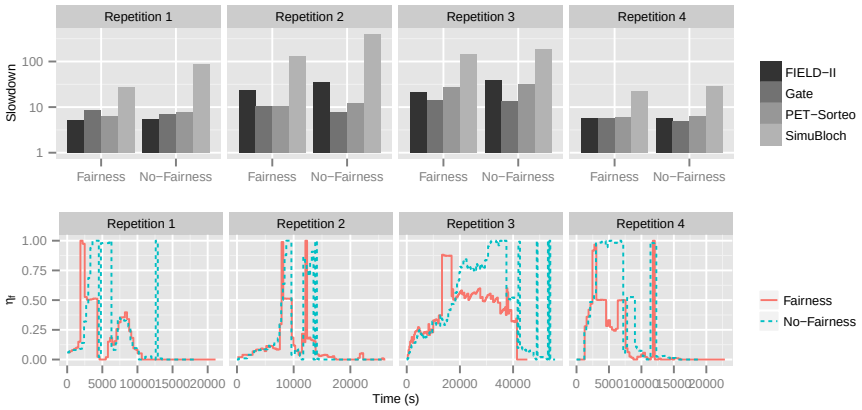
Run	Type	m (secs)	$\bar{w}$ (secs)	s
1	No-Fairness	27854	18983	196.15
	Fairness	9531	4313	38.43
2	No-Fairness	27784	19105	210.48
	Fairness	13761	10538	94.25
3	No-Fairness	14432	13579	182.68
	Fairness	9902	8145	122.25
4	No-Fairness	51664	47591	445.38
	Fairness	38630	27795	165.79

Table 3. Experiment 2: `SimuBloch`'s makespan, average wait time and slowdown.

remaining tasks. The worst-case scenario is a configuration where the number of available resources stays constant and equal to the number of tasks in the first submitted workflow: in this case, no action could be taken until the first workflow completes, and the method would not do better than first-come-first-served. Pre-emption of running tasks should be considered to address that.

4 Conclusion

We presented a method to address unfairness among workflow executions in an online and non-clairvoyant environment. We defined a novel metric η_u quantifying unfairness based on the fraction of pending work in a workflow. It compares workflow activities based on their ratio of queuing tasks, their relative durations, and the performance of resources where tasks are running. Performance is defined from the variability of task duration in the activity: good performance is assumed to lead to homogeneous task durations. To separate fair configurations from unfair ones, a threshold on η_u was determined from platform traces. Unfair configurations are handled by increasing the priority of pending tasks in the least



Repetition 1	σ_s	μ (s)
NF	40.29	4443
F	10.52	2689

Repetition 2	σ_s	μ (s)
NF	192.58	4004
F	58.83	2653

Repetition 3	σ_s	μ (s)
NF	81.81	25173
F	60.56	18537

Repetition 4	σ_s	μ (s)
NF	11.61	6613
F	8.10	3303

Fig. 4. Experiment 3 (different workflows). Top: comparison of the slowdown; middle: unfairness degree η_u; bottom: unfairness μ and slowdown standard deviation.

performing workflows. This is done by estimating the number of running tasks that these workflows should have to bring η_u under the threshold value.

The method was implemented in the MOTEUR workflow engine and deployed on EGI with the DIRAC resource manager. We tested it on four applications extracted from VIP, a science gateway for medical simulation used in production. Three experiments were conducted, to evaluate the capability of the method to improve fairness (i) on identical workflows, (ii) on workflow sets containing a very short execution and (iii) on different workflows. In all cases, results showed that our method can very significantly reduce the standard deviation of the slowdown, and the average value of our metric η_u.

The work presented here is a step in our attempt to control computing platforms where very little is known about applications and resources, and where situations change over time. Our works in [12,20] consider similar platform conditions but they target completely different problems, namely fault-tolerance and granularity control. We believe that results of this paper are the first ones presented to control fairness in such conditions which are often met in production platforms. Future work could include task pre-emption in the method, and a sensitivity analysis on the influence of the relative task duration ($T_{i,j}$) and of the performance factor ($P_{i,j}$).

5 Acknowledgment

This work is funded by the French National Agency for Research under grant ANR-09-COSI-03 “VIP”. The research leading to this publication has also received funding from the EC FP7 Programme under grant agreement 312579 ER-flow = Building an European Research Community through Interoperable Workflows and Data. Results obtained in this paper were computed on the biomed virtual organization of the Eu-

ropean Grid Infrastructure (http://www.egi.eu). We thank the European Grid Infrastructure and its supporting National Grid Initiatives, in particular France-Grilles, for providing the technical support, computing and storage facilities.

References

1. N'Takpe, T., Suter, F.: Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. IPDPS '09 (2009) 1–8
2. Zhao, H., Sakellariou, R.: Scheduling multiple DAGs onto heterogeneous systems. IPDPS'06 (2006) 159–159
3. Casanova, H., Desprez, F., Suter, F.: On cluster resource allocation for multiple parallel task graphs. J. of Par. and Dist. Computing **70**(12) (2010) 1193 – 1203
4. Glatard, T., et al.: A virtual imaging platform for multi-modality medical image simulation. IEEE Trans Med Imaging **32** (2013) 110–118
5. Shahand, S., et al.: Front-ends to Biomedical Data Analysis on Grids. In: Proceedings of HealthGrid 2011, Bristol, UK (june 2011)
6. Kacsuk, P.: P-GRADE Portal Family for Grid Infrastructures. Concurrency and Computation: Practice and Experience **23**(3) (2011) 235–245
7. Barbera, R., et al.: Supporting e-science applications on e-infrastructures: Some use cases from latin america. In: Grid Computing. (2011) 33–55
8. Hsu, C.C., Huang, K.C., Wang, F.J.: Online scheduling of workflow applications in grid environments. Fut. Gen. Computer Systems **27**(6) (2011) 860 – 870
9. Arabnejad, H., Barbosa, J.: Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In: ISPA'12. (july 2012) 633 –639
10. Sabin, G., Kochhar, G., Sadayappan, P.: Job fairness in non-preemptive job scheduling. ICPP'04 (2004) 186–194
11. Hirales-Carbajal, A., et al: Multiple workflow scheduling strategies with user run time estimates on a grid. Journal of Grid Computing **10** (2012) 325–346
12. Ferreira da Silva, R., Glatard, T., Desprez, F.: Self-healing of operational workflow incidents on distributed computing infrastructures. CCGrid'12 (2012) 318–325
13. Ferreira da Silva, R., Glatard, T.: A Science-Gateway Workload Archive to Study Pilot Jobs, User Activity, Bag of Tasks, Task Sub-Steps, and Workflow Executions. In: CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing. (2012)
14. Montagnat, J., et al.: A data-driven workflow language for grids based on array programming principles. In: WORKS'09. , Portland, USA, ACM (2009) 1–10
15. Glatard, T., et al.: Flexible and Efficient Workflow Deployment of Data-Intensive Applications on Grids with MOTEUR. IJHPCA **22**(3) (August 2008) 347–360
16. Tsaregorodtsev, A., et al.: DIRAC3. The New Generation of the LHCb Grid Software. Journal of Physics: Conference Series **219**(6) (2009) 062029
17. Jan, S., et al.: Gate v6: a major enhancement of the gate simulation platform enabling modelling of ct and radiotherapy. Phys. in Med. and Biol. **56**(4) (2011) 881–901
18. Jensen, J., Svendsen, N.B.: Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. IEEE UFFC **39**(2) (1992) 262–267
19. Reilhac, A., et al.: PET-SORTEO: Validation and Development of Database of Simulated PET Volumes. IEEE Trans. on Nuclear Science **52** (2005) 1321–1328
20. Ferreira da Silva, R., Glatard, T., Desprez, F.: On-line, non-clairvoyant optimisation of task granularity in distributed workflows. Euro-Par 2013, to appear (2013)